# Peripheral Pin Select APP -PIC32 Development

A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University
in Partial Fulfillment of the Requirements for the Degree
of
Master of Engineering, Electrical and Computer Engineering

Submitted by
Lin Wang
MEng Advisor: Bruce Land
Degree Date: January 2017

# Index

# Abstract

**Master of Engineering Program**
**School of Electrical and Computer Engineering**
**Cornell University**
**Design Project Report**

**Project Title:** Peripheral Pin Select APP
-PIC32 Development

**Author:** Lin Wang

## Abstract:

PIC32 is a 32-bit microcontroller manufactured by Microchip. Because it supports more peripherals than the number of pins it has, it has a feature called Peripheral Pin Select (PPS), which maps groups of peripherals to groups of pins by writing different values to the four least significant bits of certain registers. The large number of four-bit codes are hard to remember and it takes time to do look up each time configuration of a pin is needed. In this project, I wrote a PPS web application that does the pin assignments. After users specify pins and peripherals, the application maps peripherals to pins using user's constraints. If mapping is successful, configuration code in C language will be provided; otherwise, errors will be reported and appropriate suggestions will be provided.

**Keywords:** PIC32, PPS

## Executive Summary

PIC32 is a 32-bit microcontroller manufactured by Microchip. Because it supports more peripherals than the number of pins it has, it has a feature called Peripheral Pin Select (PPS), which maps groups of peripherals to groups of pins by writing different values into the four least significant bits of certain registers. The large number of four-bit codes are hard to remember and it takes time to do look up each time configuration of a pin is needed. In this project, I wrote a PPS web application that does the pin assignments. After users specify pins and peripherals, the application maps peripherals to pins using user's constraints. If mapping is successful, configuration code in C language (as shown in Figure 4) will be provided; otherwise, errors will be reported and appropriate suggestions (as shown in Figure 6) will be provided.

PIC32 is currently being used in Cornell ECE 4760 (Microcontroller Lab). The specific microcontroller model used in that course is PIC32MX250F128B, which has only 28 pins, include those for power, ground, etc. But it supports more than 40 peripherals (43 configurable ones as in the reference table in Figure 7).

The project contains three files: one HTML file containing all the tagged elements, one JavaScript file storing and processing all the data, and one CSS file describing how each HTML element should be represented. I wrote the program incrementally, and this report is also written following the timeline. The whole past semester is divided into six periods because I kind of reached a "milestone" at the end of each period. In the first period, I did some background research, and found that the algorithm for this project would be quite easy. In the second period, I decided to use checkboxes to gather input after evaluating against with other choices. I also decided the user interface layout and decided to use list and dictionary as main data structures. In the third period, I figured how to add even handlers using closures and wrote other supporting functions. All work done in this period is only for part of pins. In the fourth period, I scaled existing code to get a complete data structure and function for all pins. I also decided to use model-view-controller (MVC) design pattern for the program. In the fifth period, I refined the user interface by making different views hide and show at desired time. I also wrote a function to order the checkboxes alphabetically without arranging the order of the HTML elements manually. In the last period, I added one button to enable users to assign one peripheral to a specific pin. In the end, I got a completely working pin assignment web application as expected.

## Introduction

For my project, I wrote a web application to do pin assignments for user-selected peripheral combination for a PIC32 microcontroller. This project contains three files, one HTML file, one JavaScript file and one CSS file. The HTML file renders the user interface. The JavaScript file stores and processes data, handles events. And the CSS describes the presentation of the HTML elements. If the assignment is successful, the program will generate the configuration codes (in C language as shown in Figure 4) along with comments; otherwise, the program will report errors and provide appropriate suggestions (as shown in Figure 6).

## Relevant PIC32 information

PIC32 is a powerful microcontroller designed by Microchip. It is currently being used in Cornell ECE 4760 (Microcontroller Lab). The specific microcontroller model used in that course is PIC32MX250F128B, which has only 28 pins, include those for power, ground, etc. It supports more than 40 peripherals (43 configurable ones as in the reference table in Figure 7). It has a feature called Peripheral Pin Select (PPS), which maps groups of pins to groups of peripherals. Configurations of pins are done by writing different values into the four least significant bits of certain registers. The large number of 4-bit codes are almost impossible to remember, and it takes quite some time to do look up every time a pin assignment is needed. This is also the motivation of this project.

## Design

**Study PPS, do research, and decide what kind of algorithm to use**

During this period, I spent quite some time reading PIC32 family manual, and doing research online. I found the peripheral to pin multiplexing is simpler than I thought it was. I thought there were some overlapping between groups of peripherals or between groups of pins. The truth was that there was almost no overlapping, besides two exceptions, SDO1 and SDO2. These two output peripherals can be mapped to pins in pin group three and pin group four. Pin group three refers to the group of pins in row four and column three of the pin to peripheral mapping table on the reference page of the application. So the algorithm is easy: as long as the total number of input and output peripherals is smaller than the number of pins left in that group, the pin assignments can be successful. Any pin can be assigned to any peripheral of the same group. I know this may raise a question: what if users want to map a peripheral to a specific pin. I figured a solution to this problem, which will be explained later.

**Decide how to gather input**

Immediately after I started thinking about how to gather user input, checkboxes came into my mind. And I actually used them after comparing it with other choices as shown in Figure 2. I also thought about having users to enter the names of peripherals and pins in textboxes. But this would complicate the data-gathering task, and users would not necessarily like it.

**User interface layout**

I decided to sort all the checkboxes into three blocks, one block contains all the pins; one block contains all the input peripherals; the remaining block contains all the output peripherals. The pin block sits at the top horizontally, which is used to reserve occupied pins. The other two blocks follow the pin group, and are positioned side by side vertically. This UI layout is as shown in result section and never changed as shown in Figure 2.

**What data structures to use**

As I mentioned earlier, peripherals and pins were divided into four almost non-overlapping groups, so I decided to gather input and manage data in unit of group. And furthermore, as all the input were gathered with checkboxes, data structures for peripheral groups and pin group should be the same. So the sub problem was how to gather data for one group. Initially I wrote JavaScript code to gather input for the input peripherals of the first group. To gather input peripherals, three data structures were used. One list called *inputPeripheral*, which stores the names of all the input peripherals; another list named *inputChecked*, which stores the user selected input peripherals; one dictionary called *input*, which remembers the checked or unchecked state of each checkbox within its group. *inputPeripheral* was used to traverse and index into the dictionary to check if corresponding input peripheral is selected. The values of the key-value pairs in the dictionary were all initialized to -1 to indicate unselected. When a checkbox is checked, the value of corresponding key-value pair is flipped. The following are the three data structures for the input peripherals of the first group.

*input: {int4:-1, t2ck: -1, ic4:-1, ss1:-1, refclk1:-1}*
*inputPeripheral: ["int4", "t2ck", "ic4", "ss1", "refclk1"]*
*inputChecked: []*

I spent quite some time thinking about what data structure to use to store the states of the checkboxes. At first, I thought a list could be used; when one peripheral is checked, it is inserted into the list, when unchecked, it is removed from the list. This is doable, and quite straightforward. If we do not consider unchecking the checkboxes, this seems quite a good idea. But it is not practical to make this assumption. User may check a checkbox by mistake, or even change mind to use another peripheral. Considering this, list does not seem to be a good data structure, because it may require traversing the whole list to see if a peripheral is already in the list. If one peripheral is already in the list, as long as it is found, it can simply be removed from the list; if not, the whole list needs to be traversed to make sure it does not exist, in which case it will be added in the end of the list. In the case where a peripheral is already in the list, simply removing it is not enough, because a hole appears. Some operation is needed: either all items after the hole are moved one spot forward or the last item is moved to the hole. The latter operation is better because it cost less and the order of the items does not matter.

Eventually I used one dictionary to store the state, and one list to gather the checked peripherals later when user click submit button. This is better than using solely a list mentioned in previous paragraph in several perspectives. First, my implementation treats checking and unchecking operation exactly the same, requiring simply flipping the value. On the contrary, with one list, the two operation were treated differently: when checked, a peripheral needs to be added to the list; when unchecked, it needs to be removed. And the program needs to traverse the list to decide if the peripheral needs to be added to or removed from the list. My implementation does not even care if current operation is checking or unchecking. Moreover, my implementation should be more efficient. I am not sure if a dictionary is more efficient than a list. A list requires traversal to add or remove an item. A dictionary may also require because key-value pair are stored in “random” order, even though we do not need to do searching on our own like what we do for a list. Even if dictionary is not more efficient than a list, my implementation should still be more efficient because it does not require rearranging items when a peripheral is unchecked.

Actually, later when I almost finished the project I found out that I did not need to remember the state of the checkboxes at all. JQuery has built-in method to get this info. So actually, I could have made the program simpler by removing all the dictionaries and removing all the event handlers associated with each checkboxes. But my implementation is fine, so I decide to leave it as it is. So in next session I will still talk about those even handlers.

**Adding event handlers, closure**

Debugging JavaScript code is harder than programs in other languages I used before, like C or C++. Browsers do not report errors when running JS files. So I have to write the program little by little, and test each basic block before continuing writing next one. This time I can definitely say I used a test-driven and incremental design approach to write the code, because otherwise I would have gone into trouble for too many times.

JQuerry was used to add event handlers. After testing on one peripheral, I added event handlers for all the input peripherals in the first group. Closure was used to add event handlers, and the code in the following contains a closure. Closure is a nested function that has access to the environment where it was defined. I learnt this kind of function in doing this project. My main programming language was C++ and C, and there is no such kind of function in them. JavaScript, Python and some other languages support closures and high order functions because they use different scoping mechanism than C and C++. Per my understanding, C and C++ use dynamic scope, while Python and JavaScript use lexical scope. In languages with lexical scope, name resolution depends on the location in the source code and lexical context. In contrast, in language with dynamic scope, the name resolution depends upon the program state determined by the execution context or calling context. This sound confusing, in next paragraph I will explain by analyzing that piece of code as shown in the following.

```
$("#"+ temp.inputPeripheral[j]).click((function(name, array){
            return function(){
                  controller.flip(name, array);
            };
})(temp.inputPeripheral[j], temp.input));
```

The code is in JQuerry. *$("#"+ temp.inputPeripheral[j])* is JQuerry's selector syntax. This statement select the HTML element with id *temp.inputPeripheral[j]. click()* method attaches an even handler function to that HTML element. The even handler function is what is inside the parenthesis, which is a closure. The outer function of the closure executes only once, and *temp.inputPeripheral[j], temp.input* are substituted into the formal parameters *name* and *array*. After the execution of the outer function, a function is returned. And the function execute *controller.flip(temp.inputPeripheral[j], temp.input).* From this point on, any time the checkbox with id *temp.inputPeripheral[j]* is clicked, *controller.flip(temp.inputPeripheral[j], temp.input)* will be executed. This is why JavaScript is lexical scoped: *controller.flip()* holds a handle to variables *temp.inputPeripheral[j], temp.input),* which are variables passes into its parent function. The resolution of *name* and *array* in *controller.flip()* is completely determined by the source code, not its execution context.

**Other functions**

After the event handlers are added, the rest of the code are easy to write. Three additional functions were written. *getCheckedPeripheral()* sorts all the checked peripherals into the second list mentioned earlier. It knows if a peripheral is checked by checking the value of corresponding key-value pair in the dictionary, 1 for checked and -1 for unchecked. *checkForError()* checks to see if errors exist. Error simply means the number of pins left is smaller than the number of peripherals selected. *doAssignment()* calls *checkForError()* in the beginning, if errors exist, it reports the errors and returns; otherwise, it does the assignment, and outputs the configuration code.

**Complete data structure**

Later I found that when I wrote the program for the first group, the variables I created made it very difficult to scale the program easily. The variable holding the names of all input peripherals in the first group was g1input (short of group one input) instead of input. Using this naming convention, I have to create g2input, g3input and so on if I were to write code for the other groups as well. I soon realized that this was not a good practice, even though I can still get the program to work. And including digit 1 in an identifier is itself not a good practice because 1 (digit) and l (letter) are indistinguishable in many systems. So I did some research online, searching for example data structures in JavaScript. I learnt that in JavaScript, dictionary (or object) can be quite complex. Eventually, I got a data model like the one in the following. Here I take only part of my data structure as an example.

```
var model = {
	message: "",
	groups: [
		//group 0
		{
			input: {int4:-1, t2ck: -1, ic4:-1, ss1:-1, refclk1:-1},
			inputPeripheral: ["int4", "t2ck", "ic4", "ss1", "refclk1"],
			inputChecked: [],
			output: {u1tx:-1, u2rts_n:-1, ss1_o:-1, oc1:-1, c2out:-1},
			outputPeripheral: ["u1tx", "u2rts_n", "ss1_o", "oc1", "c2out"],
			outputPeripheralValue: {u1tx:"0x0001", u2rts_n:"0x0002",
			ss1_o:"0x0003", oc1:"0x0005", c2out:"0x0007"},
			outputChecked: [],
			pin: {pin2: 1, pin7: 1, pin11: 1, pin16: 1, pin26: 1},
			pinName: ["pin2", "pin7", "pin11", "pin16", "pin26"],
			pinReg: {pin2:"RPA0", pin7:"RPB3", pin11:"RPB4", pin16:"RPB7",
			pin26:"RPB15"},
			pinValue: {pin2:"0x0000", pin7:"0x0001", pin11:"0x0002",
			pin16:"0x0004", pin26:"0x0003"},
			pinLeft: []
		},
		//group 1
		{
			input: {int3: -1, t3ck: -1, ic3: -1, u1cts: -1, u2rx: -1, sdi1: -1},
			inputPeripheral: ["int3", "t3ck", "ic3", "u1cts", "u2rx", "sdi1"],
			inputChecked: [],
			output: {oc2: -1, c3out: -1},
			outputPeripheral: ["oc2", "c3out"],
			outputPeripheralValue: {oc2: "0x0005", c3out: "0x0007"},
			outputChecked: [],
			pin: {pin3: 1, pin5: 1, pin14: 1, pin17: 1, pin22: 1},
			pinName: ["pin3", "pin5", "pin14", "pin17", "pin22"],
			pinReg: {pin3: "RPA1", pin5: "RPB1", pin14: "PRB5", pin17: "RPB8",
			pin22: "RPB11"},
			pinValue: {pin3: "0x0000", pin5: "0x0002", pin14: "0x0001", pin17:
			"0x0004", pin22: "0x0003"},
			pinLeft: []
		}
}
```

Complex data structures turned out to be very handy and powerful. All the data in my program are stored in the above object called model. If I want to get all the input peripheral names of group i, *model.groups[i].inputPeripheral* gives me the result. With this structure, many repetitive work can be done using for loops, which is impossible for the c-style naming convention I used initially. Before doing this project, I knew JavaScript was not as energy efficient as C, so I thought C was superior. Now I realize that I was wrong. JavaScript has its own strengths, it is more versatile, and more powerful in its own way.

**JavaScript file design pattern**

While doing research online, I also learnt a design pattern: model-view-controller (MVC). This is a widely-used design pattern in web development. It separates a program into three part: model, view and controller. Model directly stores and manages the data. A view can be any output representation of the information. And the controller accepts input and convert it to commands. One essential idea of MVC is controller never directly talks with view. Figure 1 is a UML-like diagram showing my JavaScript file design pattern. It seems that I did not follow exactly the MVC pattern, but I think my program is quite neat and easy to follow.
My JavaScript file contains three objects named *model, pinSelectionView and controller*. *Model* stores all the data. *pinSelectionView* adds all event handlers and describes how the HTML component should be displayed. The *controller* manipulates the data structures.

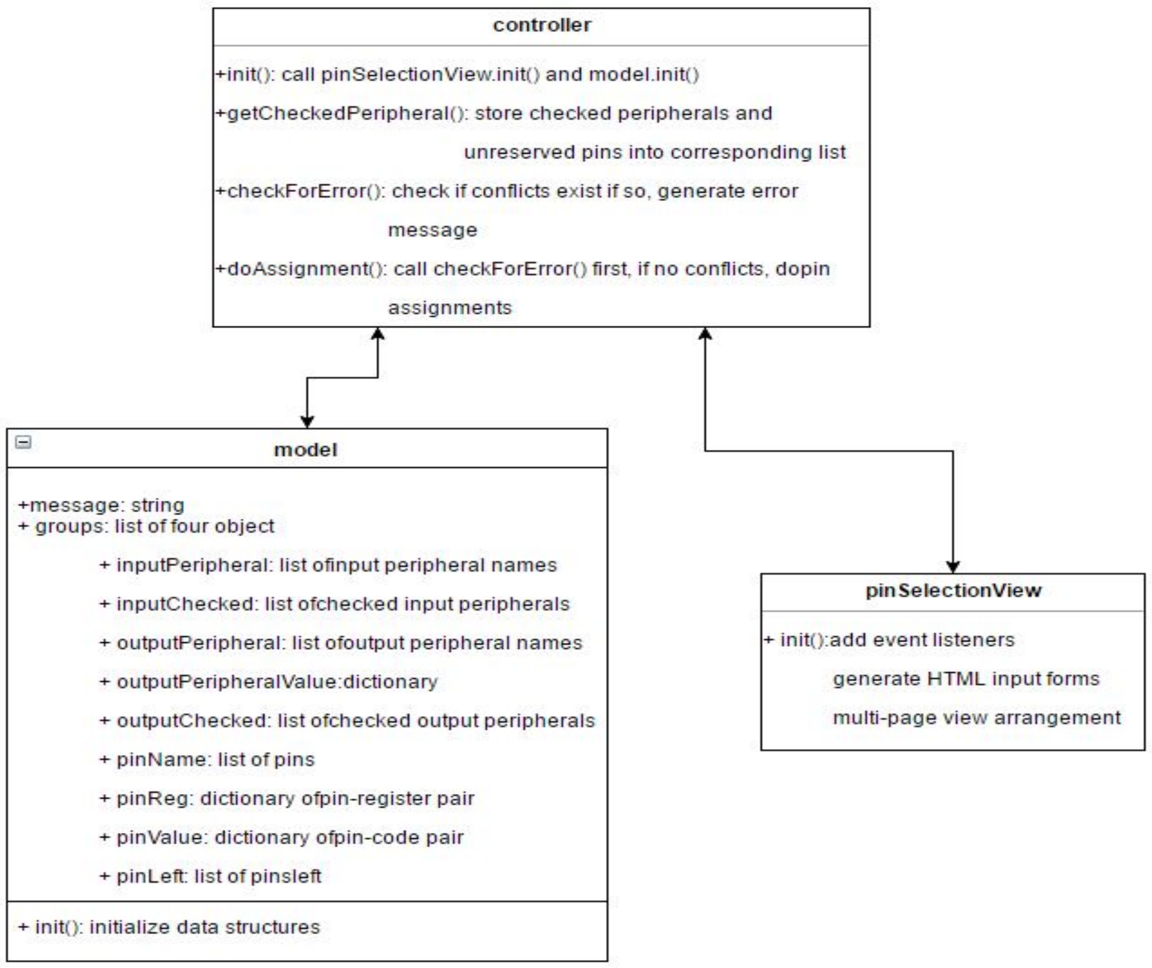

controller
+init(): call pinSelectionView.init() and model.init()
+getCheckedPeripheral(): store checked peripherals and unreserved pins into corresponding list
+checkForError(): check if conflicts exist if so, generate error message
+doAssignment(): call checkForError() first, if no conflicts, dopin assignments
model
+message: string
+ groups: list of four object
+ inputPeripheral: list ofinput peripheral names
+ inputChecked: list ofchecked input peripherals
+ outputPeripheral: list ofoutput peripheral names
+ outputPeripheralValue:dictionary
+ outputChecked: list ofchecked output peripherals
+ pinName: list of pins
+ pinReg: dictionary ofpin-register pair
+ pinValue: dictionary ofpin-code pair
+ pinLeft: list of pinsleft
+ init(): initialize data structures
pinSelectionView
+ init():add event listeners
generate HTML input forms
multi-page view arrangement

**Figure 1. JavaScript file design pattern (MVC)**

**Dealing with hiding and showing of different views**

Up to this point, all the HTML elements in my program appeared on the same page from top down. As I was to develop a web app, I thought it would be better if I could separate pin selection, output and reference into three views that can be hidden or shown with a click of a button. This way each of the three views can fit into one screen, and users will never need to scroll up and down when using the application. Users only need one type of operation to navigate through the application, clicking the left button of their mouse. And after all, they need to click on checkboxes to do pin and peripheral selection. The hiding and showing of HTML elements can be easily achieved using JQuerry. The two main methods for this task are shown below. The first function is called by the event handler function in the second block, which hides all the views first and then shows the view with the name passed in. The second block adds event handlers to appropriate HTML element. To achieve the hiding and showing function, some other modification in the HTML file are also needed. I will not explain them in details here.

```
function showView(viewName) {
        $('.view').hide();
        $('#' + viewName).show();
}

$('[data-launch-view]').click(function (e) {
        e.preventDefault();
        var viewName = $(this).attr('data-launch-view');
        showView(viewName);
});
```

**Ordering checkboxes alphabetically**

When I wrote of the HTML file, for the sake of convenience I grouped peripherals into several groups. But later I was suggested by my advisor to order them alphabetically. The easiest way is to rearrange the checkbox elements manually. But this would be tedious and makes the program hard to maintain, even though no one would come back to add a peripheral to any group. What I did was keep the HTML as it was, but added a sorting function in the JavaScript file to order the checkbox elements. That function is as follows.

```
function sortUnorderedList(ul, sortDescending, nameOfForm) {
        if (typeof ul == "string") ul = document.getElementById(nameOfForm);

        var lis = $("#" + nameOfForm+ " label");
        var vals = [];

        for (var i = 0, l = lis.length; i < l; i++)
                vals.push(lis[i].innerHTML);
        vals.sort();
        if (sortDescending) vals.reverse();
        for (var i = 0, l = lis.length; i < l; i++)
                lis[i].innerHTML = vals[i];
}
```

This function first sorts innerHTML of all the elements into a list. Then it orders the items of the list alphabetically by calling JavaScript's built-in *sort()* function. In the end it assigns the ordered innerHTML items to the original checkbox elements to achieve the purpose of ordering. This function decouples how HTML elements are represented on the screen from how HTML elements are organized in the codes. This would save a lot of time if later the program needs modification.

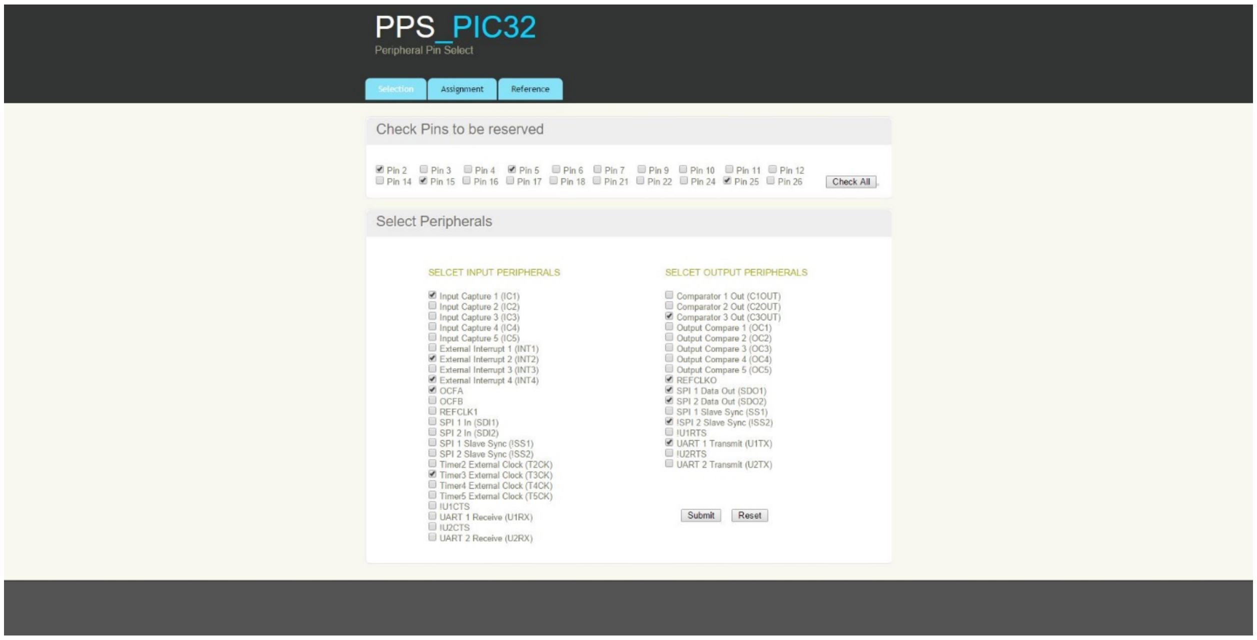
In the end of this period, I wrote the CSS file to beautify the user interface. There is nothing special about this file, it only involves some tedious adjustments to a large number of parameters.

### Dealing with the mapping one peripheral to one specific pin problem

As I mentioned earlier, the pin assignments are kind of a random process. I also raised a question: what if users want to map a peripheral to a specific pin. To solve this problem, I thought about adding a dropdown list to the end of each of the input and output peripheral checkbox. The items in a dropdown list would be all the possible pins the corresponding peripheral can be mapped to. This could achieve the goal but it would make the UI crowded and ugly to look at. And this also seems to defeat the purpose of this pin assignment application. Users want the program to do the job for them. They would not want to map the peripherals one by one very often. I later came up with a solution that requires adding only one button to the program. One button would not make the UI crowded, at the same time it enables users to assign one peripheral to a specific pin. The button is used to check all the pins. With this button, whenever a user wants to map a peripheral A to a pin B, he or she simply clicks Check All button to reserve all pins, uncheck pin B to make it the only pin that is available, check peripheral A, and then click submit button.

## User interface and results

This section provides some screenshots of the UI, output and the reference page of the application. Figure 2 shows the complete view of the user interface, while Figure 3 through Figure 7 only show the central part of corresponding view with blank space on both left and right side excluded. Figure 3 shows a possible-to-map selection in which four pins are reserved, five input peripherals and six output peripherals are selected. Figure 4 shows the configuration code in C language corresponding to the selection in Figure 3. Users can simply copy and paste the code into their program. Figure 5 shows an impossible-to-map selection because for some groups number of peripherals selected is larger than number of pins available. Figure 6 shown the error message and suggestions corresponding to selection in Figure 5. Figure 7 shows the reference page that contains one peripheral to pin mapping table and one PIC32 pin layout diagram. In the table, any peripheral (input or output peripherals) can be mapped to any pin in the same row. From the table, it can also be verified SDO1 and SDO2 are the only two peripherals that can be mapped to more than one group of pins, two groups to be specific. The pin layout diagram shows what peripherals each pin can be mapped to. As shown in these figures, the application has a clean and easy-to-use user interface.



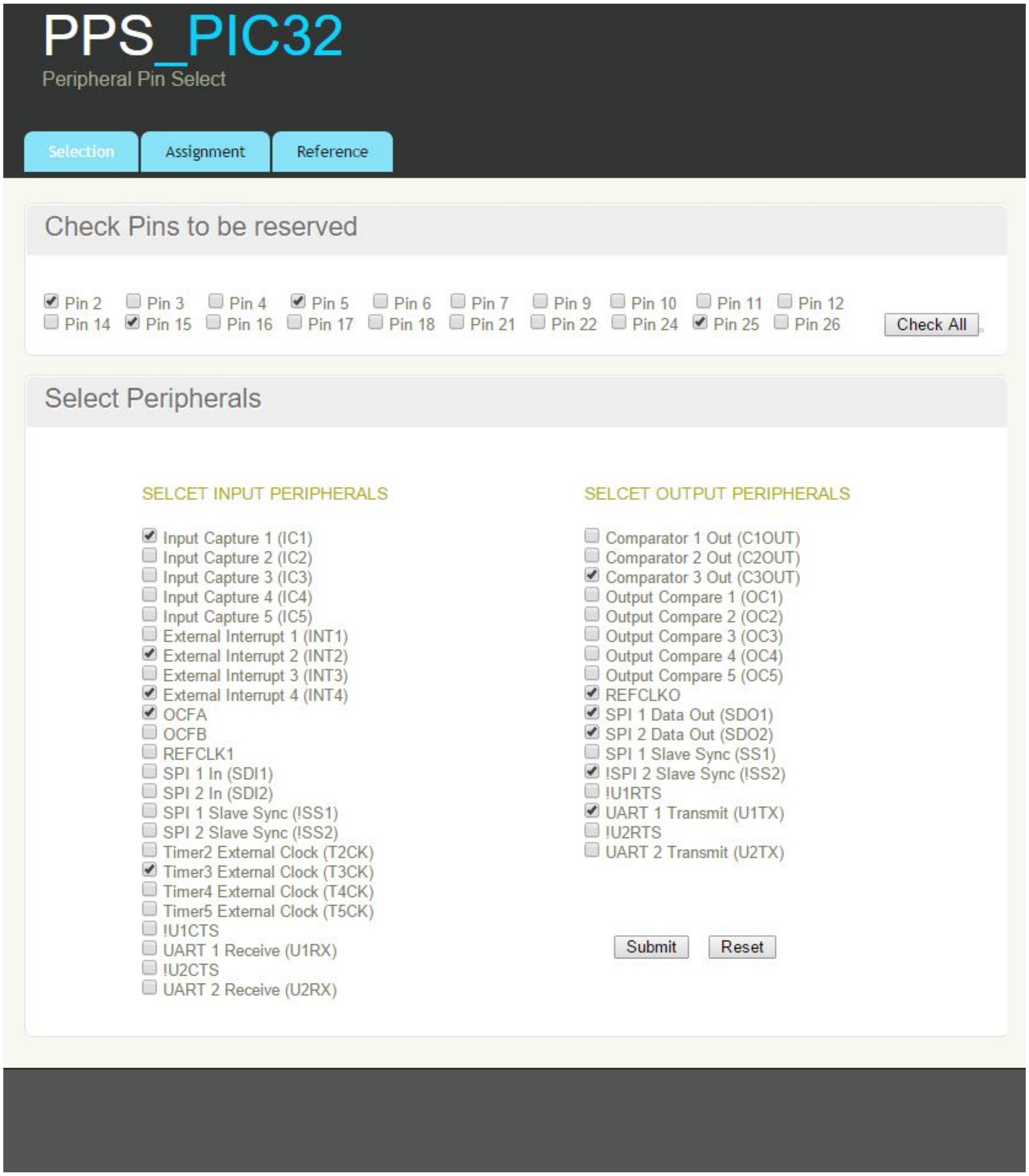
Check Pins to be reserved

[✓ Pin 2] Pin 3 Pin 4 [✓ Pin 5] Pin 6 Pin 7 Pin 9 Pin 10 Pin 11 Pin 12
Pin 14 [✓ Pin 15] Pin 16 Pin 17 Pin 18 Pin 21 Pin 22 Pin 24 [✓ Pin 25] Pin 26

Select Peripherals

| SELCET INPUT PERIPHERALS | SELCET OUTPUT PERIPHERALS |
|---|---|
| [✓ Input Capture 1 (IC1)] | Comparator 1 Out (C1OUT) |
| Input Capture 2 (IC2) | Comparator 2 Out (C2OUT) |
| Input Capture 3 (IC3) | [✓ Comparator 3 Out (C3OUT)] |
| Input Capture 4 (IC4) | Output Compare 1 (OC1) |
| Input Capture 5 (IC5) | Output Compare 2 (OC2) |
| External Interrupt 1 (INT1) | Output Compare 3 (OC3) |
| [✓ External Interrupt 2 (INT2)] | Output Compare 4 (OC4) |
| External Interrupt 3 (INT3) | Output Compare 5 (OC5) |
| [✓ External Interrupt 4 (INT4)] | [✓ REFCLKO] |
| [✓ OCFA] | [✓ SPI 1 Data Out (SDO1)] |
| OCFB | [✓ SPI 2 Data Out (SDO2)] |
| REFCLK1 | SPI 1 Slave Sync (SS1) |
| SPI 1 In (SDI1) | [✓ !SPI 2 Slave Sync (!SS2)] |
| SPI 2 In (SDI2) | !U1RTS |
| SPI 1 Slave Sync (!SS1) | [✓ UART 1 Transmit (U1TX)] |
| SPI 2 Slave Sync (!SS2) | !U2RTS |
| Timer2 External Clock (T2CK) | UART 2 Transmit (U2TX) |
| [✓ Timer3 External Clock (T3CK)] | |
| Timer4 External Clock (T4CK) | |
| Timer5 External Clock (T5CK) | |
| !U1CTS | |
| UART 1 Receive (U1RX) | |
| !U2CTS | |
| UART 2 Receive (U2RX) | |

**Figure 2. User interface**



Check Pins to be reserved

[✓] Pin 2 [ ] Pin 3 [ ] Pin 4 [✓] Pin 5 [ ] Pin 6 [ ] Pin 7 [ ] Pin 9 [ ] Pin 10 [ ] Pin 11 [ ] Pin 12
[ ] Pin 14 [✓] Pin 15 [ ] Pin 16 [ ] Pin 17 [ ] Pin 18 [ ] Pin 21 [ ] Pin 22 [ ] Pin 24 [✓] Pin 25 [ ] Pin 26

Select Peripherals

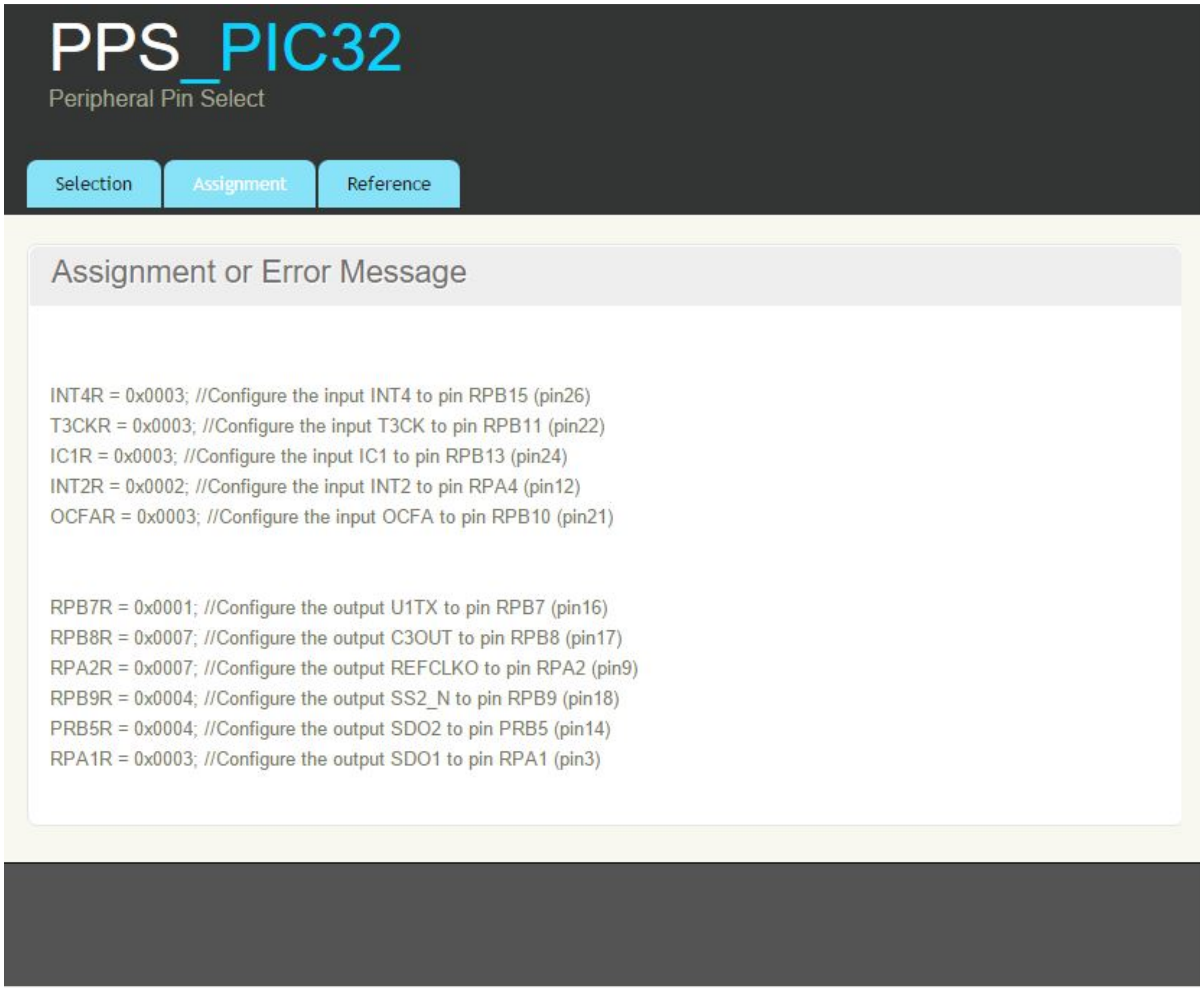
SELCET INPUT PERIPHERALS

[✓] Input Capture 1 (IC1)
[ ] Input Capture 2 (IC2)
[ ] Input Capture 3 (IC3)
[ ] Input Capture 4 (IC4)
[ ] Input Capture 5 (IC5)
[ ] External Interrupt 1 (INT1)
[✓] External Interrupt 2 (INT2)
[ ] External Interrupt 3 (INT3)
[✓] External Interrupt 4 (INT4)
[✓] OCFA
[ ] OCFB
[ ] REFCLK1
[ ] SPI 1 In (SDI1)
[ ] SPI 2 In (SDI2)
[ ] SPI 1 Slave Sync (!SS1)
[ ] SPI 2 Slave Sync (!SS2)
[ ] Timer2 External Clock (T2CK)
[✓] Timer3 External Clock (T3CK)
[ ] Timer4 External Clock (T4CK)
[ ] Timer5 External Clock (T5CK)
[ ] !U1CTS
[ ] UART 1 Receive (U1RX)
[ ] !U2CTS
[ ] UART 2 Receive (U2RX)

SELCET OUTPUT PERIPHERALS

[ ] Comparator 1 Out (C1OUT)
[ ] Comparator 2 Out (C2OUT)
[✓] Comparator 3 Out (C3OUT)
[ ] Output Compare 1 (OC1)
[ ] Output Compare 2 (OC2)
[ ] Output Compare 3 (OC3)
[ ] Output Compare 4 (OC4)
[ ] Output Compare 5 (OC5)
[✓] REFCLKO
[✓] SPI 1 Data Out (SDO1)
[✓] SPI 2 Data Out (SDO2)
[ ] SPI 1 Slave Sync (SS1)
[✓] !SPI 2 Slave Sync (!SS2)
[ ] !U1RTS
[✓] UART 1 Transmit (U1TX)
[ ] !U2RTS
[ ] UART 2 Transmit (U2TX)

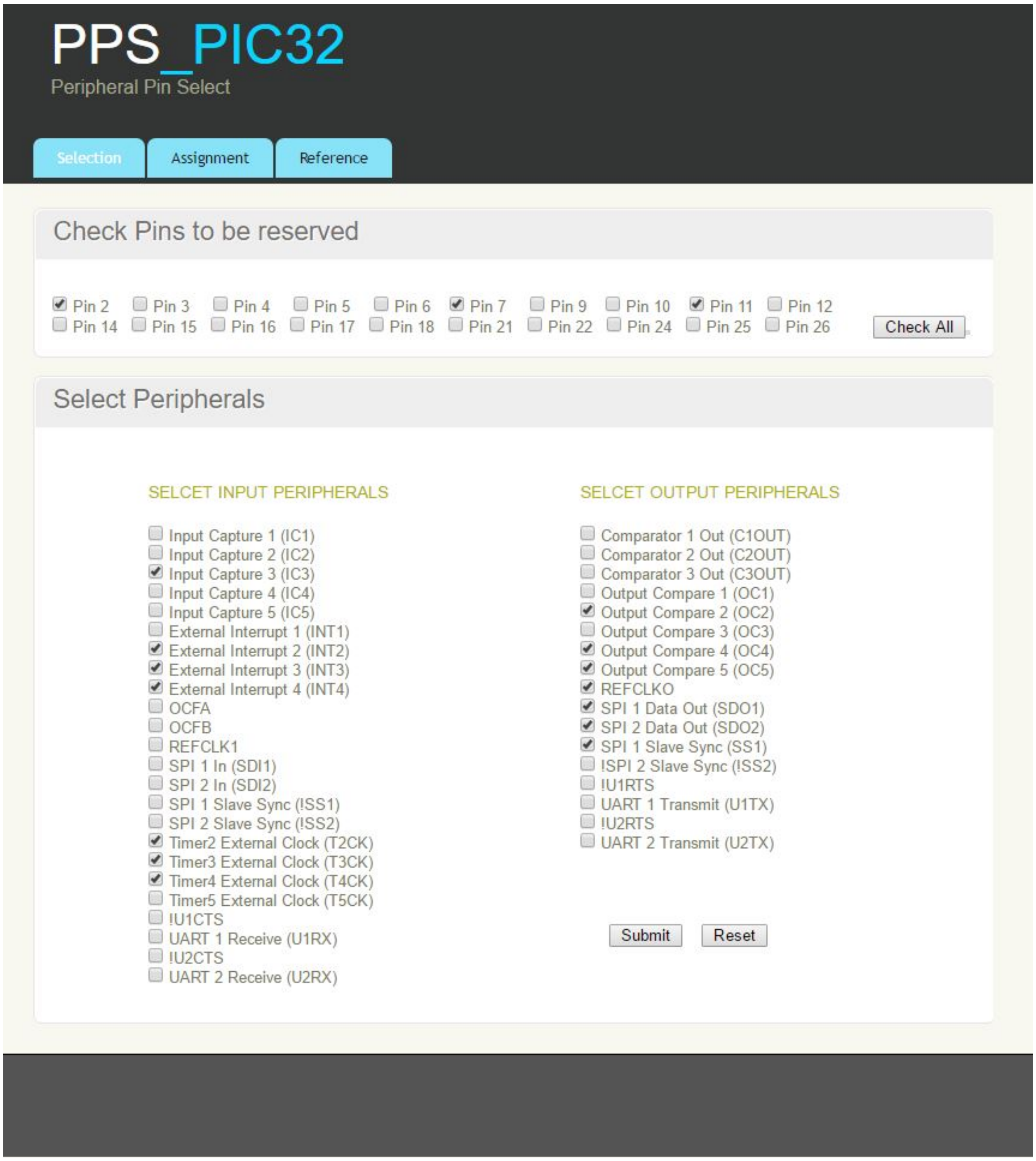
**Figure 3. 4 pins reserved, 5 input peripherals and 6 output peripherals selected (corresponding configuration code shown in Figure 4)**



```
INT4R = 0x0003; //Configure the input INT4 to pin RPB15 (pin26)
T3CKR = 0x0003; //Configure the input T3CK to pin RPB11 (pin22)
IC1R = 0x0003; //Configure the input IC1 to pin RPB13 (pin24)
INT2R = 0x0002; //Configure the input INT2 to pin RPA4 (pin12)
OCFAR = 0x0003; //Configure the input OCFA to pin RPB10 (pin21)


RPB7R = 0x0001; //Configure the output U1TX to pin RPB7 (pin16)
RPB8R = 0x0007; //Configure the output C3OUT to pin RPB8 (pin17)
RPA2R = 0x0007; //Configure the output REFCLKO to pin RPA2 (pin9)
RPB9R = 0x0004; //Configure the output SS2_N to pin RPB9 (pin18)
PRB5R = 0x0004; //Configure the output SDO2 to pin PRB5 (pin14)
RPA1R = 0x0003; //Configure the output SDO1 to pin RPA1 (pin3)
```

**Figure 4. Sample pin assignment output (for selection in Figure 3)**



Check Pins to be reserved

[✓ Pin 2] Pin 3 Pin 4 Pin 5 Pin 6 [✓ Pin 7] Pin 9 Pin 10 [✓ Pin 11] Pin 12
Pin 14 Pin 15 Pin 16 Pin 17 Pin 18 Pin 21 Pin 22 Pin 24 Pin 25 Pin 26

Select Peripherals

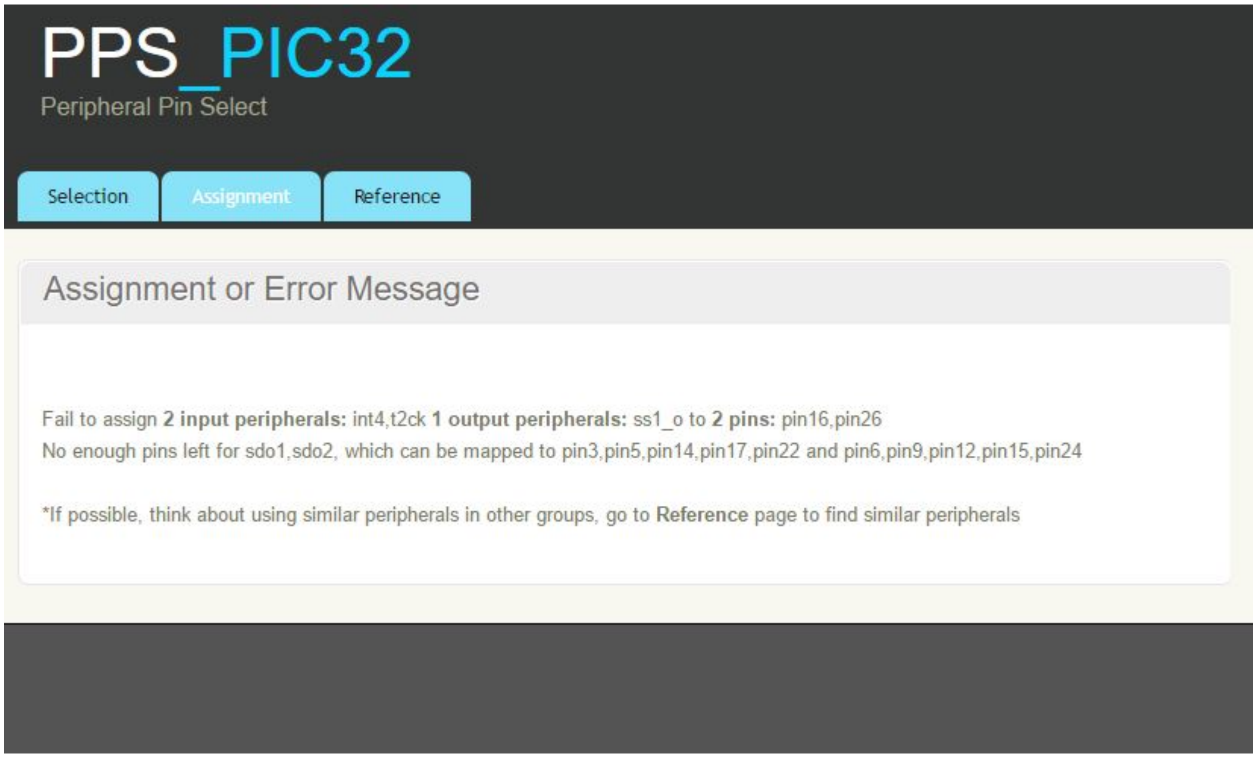
SELCET INPUT PERIPHERALS

- Input Capture 1 (IC1)
- Input Capture 2 (IC2)
- [✓ Input Capture 3 (IC3)]
- Input Capture 4 (IC4)
- Input Capture 5 (IC5)
- External Interrupt 1 (INT1)
- [✓ External Interrupt 2 (INT2)]
- [✓ External Interrupt 3 (INT3)]
- [✓ External Interrupt 4 (INT4)]
- OCFA
- OCFB
- REFCLK1
- SPI 1 In (SDI1)
- SPI 2 In (SDI2)
- SPI 1 Slave Sync (!SS1)
- SPI 2 Slave Sync (!SS2)
- [✓ Timer2 External Clock (T2CK)]
- [✓ Timer3 External Clock (T3CK)]
- [✓ Timer4 External Clock (T4CK)]
- Timer5 External Clock (T5CK)
- !U1CTS
- UART 1 Receive (U1RX)
- !U2CTS
- UART 2 Receive (U2RX)

SELCET OUTPUT PERIPHERALS

- Comparator 1 Out (C1OUT)
- Comparator 2 Out (C2OUT)
- Comparator 3 Out (C3OUT)
- Output Compare 1 (OC1)
- [✓ Output Compare 2 (OC2)]
- Output Compare 3 (OC3)
- [✓ Output Compare 4 (OC4)]
- [✓ Output Compare 5 (OC5)]
- [✓ REFCLKO]
- [✓ SPI 1 Data Out (SDO1)]
- [✓ SPI 2 Data Out (SDO2)]
- [✓ SPI 1 Slave Sync (SS1)]
- !SPI 2 Slave Sync (!SS2)
- !U1RTS
- UART 1 Transmit (U1TX)
- !U2RTS
- UART 2 Transmit (U2TX)

Figure 5. Sample impossible-to-map selection (error message shown in Figure 6)

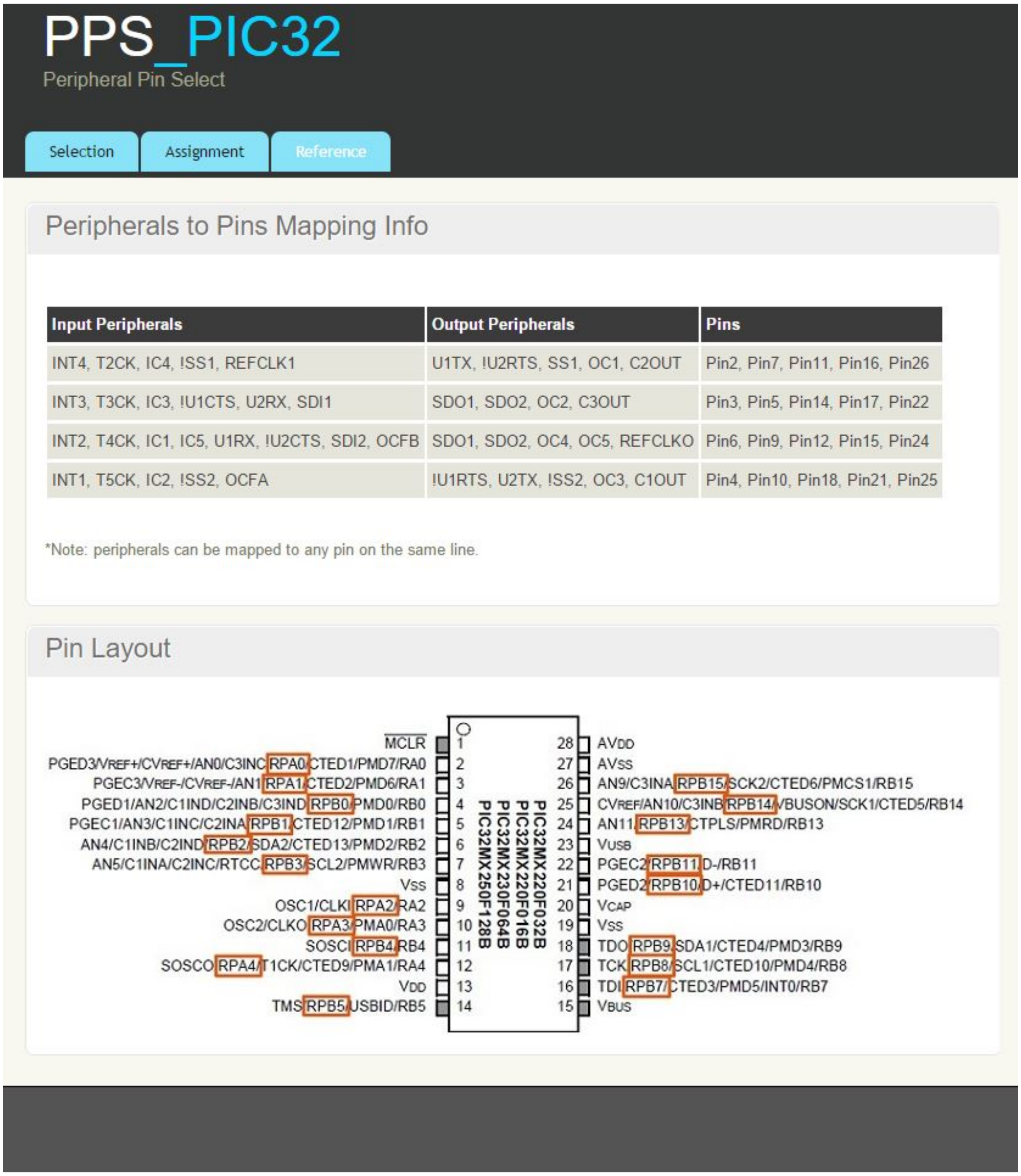


Assignment or Error Message

Fail to assign **2 input peripherals:** int4,t2ck **1 output peripherals:** ss1_o to **2 pins:** pin16,pin26
No enough pins left for sdo1,sdo2, which can be mapped to pin3,pin5,pin14,pin17,pin22 and pin6,pin9,pin12,pin15,pin24

*If possible, think about using similar peripherals in other groups, go to **Reference** page to find similar peripherals

**Figure 6. Sample error message (for selection in Figure 5)**



PPS_PIC32

Peripheral Pin Select

Peripherals to Pins Mapping Info

| Input Peripherals | Output Peripherals | Pins |
|---|---|---|
| INT4, T2CK, IC4, !SS1, REFCLK1 | U1TX, !U2RTS, SS1, OC1, C2OUT | Pin2, Pin7, Pin11, Pin16, Pin26 |
| INT3, T3CK, IC3, !U1CTS, U2RX, SDI1 | SDO1, SDO2, OC2, C3OUT | Pin3, Pin5, Pin14, Pin17, Pin22 |
| INT2, T4CK, IC1, IC5, U1RX, !U2CTS, SDI2, OCFB | SDO1, SDO2, OC4, OC5, REFCLKO | Pin6, Pin9, Pin12, Pin15, Pin24 |
| INT1, T5CK, IC2, !SS2, OCFA | !U1RTS, U2TX, !SS2, OC3, C1OUT | Pin4, Pin10, Pin18, Pin21, Pin25 |

*Note: peripherals can be mapped to any pin on the same line.

Pin Layout

MCLR 1
PGED3/VREF+/CVREF+/AN0/C3INC/RPA0/CTED1/PMD7/RA0 2
PGEC3/VREF-/CVREF-/AN1/RPA1/CTED2/PMD6/RA1 3
PGED1/AN2/C1IND/C2INB/C3IND/RPB0/PMD0/RB0 4
PGEC1/AN3/C1INC/C2INA/RPB1/CTED12/PMD1/RB1 5
AN4/C1INB/C2IND/RPB2/SDA2/CTED13/PMD2/RB2 6
AN5/C1INA/C2INC/RTCC/RPB3/SCL2/PMWR/RB3 7
VSS 8
OSC1/CLKI/RPA2/RA2 9
OSC2/CLKO/RPA3/PMA0/RA3 10
SOSCI/RPB4/RB4 11
SOSCO/RPA4/T1CK/CTED9/PMA1/RA4 12
VDD 13
TMS/RPB5/USBID/RB5 14
28 AVDD
27 AVSS
26 AN9/C3INA/RPB15/SCK2/CTED6/PMCS1/RB15
25 CVREF/AN10/C3INB/RPB14/VBUSON/SCK1/CTED5/RB14
24 AN11/RPB13/CTPLS/PMRD/RB13
23 VUSB
22 PGEC2/RPB11/D-/RB11
21 PGED2/RPB10/D+/CTED11/RB10
20 VCAP
19 VSS
18 TDO/RPB9/SDA1/CTED4/PMD3/RB9
17 TCK/RPB8/SCL1/CTED10/PMD4/RB8
16 TDI/RPB7/CTED3/PMD5/INT0/RB7
15 VBUS
PIC32MX220F032B
PIC32MX220F016B
PIC32MX230F064B
PIC32MX250F128B

**Figure 7. Reference page**

## Conclusions

As expected, I get a working pin assignment web application for a PIC32 microcontroller. If the assignment is successful, the program will output the configuration codes as shown in Figure 4; otherwise, it will report the errors to users as shown in Figure 6. This application has a user-friendly UI, and is compatible with any browser on the market. In doing this project, I learnt JavaScript, my first scripting programming language. What I learnt the most is in the designing process, in which there were still quite some tradeoffs for me to make. There was the tradeoff between memory usage and program performance, and there was the tradeoff between application usability and more features but crowded user interface. In dealing with those tradeoffs I learnt how to evaluate conflicting choices and make good decisions.

## Acknowledgements

Thanks to my advisor Bruce Land for guiding me through this designing process!

## References

http://umassamherstm5.org/tech-tutorials/pic32-tutorials/pic32mx220-tutorials/peripheral-pin-select-pps

## Appendix: User's manual

**Explain three views of the UI**

The User interface has three views. The first view (as shown in Figure 2, Figure 3 and Figure 5) is the pin selection view which can be shown by clicking the selection button in the navigation. This view contains three groups of checkboxes. The first group located at top contains checkboxes for all configurable pins and a button called *Check All*. The second group located on lower left side contains checkboxes for all input peripherals. The third group located on lower right side contains checkboxes for all output peripherals. Within each group checkboxes are ordered alphabetically. Following the third group, there are two buttons, submit button and reset button. Submit button takes user to the output view and should be used when users are done with selecting all desired peripherals. Reset button deselects all checkboxes and clears all output.

The second view (as shown in Figure 4 and Figure 6) is the output view which displays the output or error message depending on whether the assignment is successful. It can be shown by clicking the Assignment button in the navigation. And it also shows automatically after users click submit button.

The third view (as shown in Figure 7) is the reference page view, which can be shown by clicking the Reference button in the navigation. This view contains one table that shown peripheral to pin mapping information, and one pin layout diagram. In the table, any peripheral (input or output peripherals) can map to any pin in the same row. From the table, it can also be verified SDO1 and SDO2 are the only two peripherals that can be mapped to more than one group of pins, two to be specific. The pin layout diagram shown what peripherals each pin can be mapped to.

### How to do pin assignments

Open the application, reserve pins if needed, check all peripherals you want to use and then click submit button. If in the process, you want reset the program, click reset button or the refresh button of the browser.

### How to assign one peripheral to one specific pin

Open the application, click Check All button to check all pins, uncheck the pin you want to map your peripheral to, check the peripheral you need and then click submit button.

# Codes

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## HTML file

```
<!DOCTYPE html>
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>PIC32 Peripheral Pin Select</title>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" id="theme" href="pps.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
</head>

<body>
  <div id="main">
    <header>
      <div id="logo">
        <div id="logo_text">
          <!-- class="logo_colour", allows you to change the colour of the text -->
          <h1><a>PPS<span class="logo_colour">_PIC32</span></a></h1>
```

```
                    <h2>Peripheral Pin Select</h2>
        </div>
      </div>
      <nav>
        <ul class="sf-menu" id="nav">
          <li  class="current"><a data-launch-view="site_content1">Selection</a></li>
          <li id="assignment"><a data-launch-view="site_content2">Assignment</a></li>
          <li><a data-launch-view="site_content3">Reference</a></li>
        </ul>
      </nav>
    </header>
    <div class="view" id="site_content1">
      <div class="content">
        <h1>Check Pins to be reserved</h1>
        <div class="content_item">
                                        <form id="pinForm">

                    <input type="checkbox" id="pin2" >  Pin 2&nbsp&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin3" >  Pin 3&nbsp&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin4" >  Pin 4&nbsp&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin5" >  Pin 5&nbsp&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin6" >  Pin 6&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin7" >  Pin 7&nbsp&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin9" >  Pin 9&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin10"> Pin 10&nbsp&nbsp&nbsp
                    <input type="checkbox" id="pin11"> Pin 11&nbsp&nbsp
                    <input type="checkbox" id="pin12"> Pin 12&nbsp&nbsp<br />
                    <input type="checkbox" id="pin14"> Pin 14&nbsp&nbsp
                    <input type="checkbox" id="pin15"> Pin 15&nbsp&nbsp
                    <input type="checkbox" id="pin16"> Pin 16&nbsp&nbsp
                    <input type="checkbox" id="pin17"> Pin 17&nbsp&nbsp
                    <input type="checkbox" id="pin18"> Pin 18&nbsp&nbsp
                    <input type="checkbox" id="pin21"> Pin 21&nbsp&nbsp
                    <input type="checkbox" id="pin22"> Pin 22&nbsp&nbsp
                    <input type="checkbox" id="pin24"> Pin 24&nbsp&nbsp
                    <input type="checkbox" id="pin25"> Pin 25&nbsp&nbsp
                    <input type="checkbox" id="pin26"> Pin 26&nbsp&nbsp
                    &nbsp &nbsp &nbsp <button type="button" id="checkAllPins"> &nbsp Check
All &nbsp <button>
                    </form>

                    </div>
      </div>
      <div class="content">
        <h1>Select Peripherals</h1>
        <div class="content_item">
                    <form id="inputForm">
                    <h4>Selcet input peripherals</h4><br />
                    <label><input type="checkbox"   id="int4"   >  External Interrupt 4 (INT4) <br
/>  </label>
                    <label><input type="checkbox"   id="t2ck"   >  Timer2 External Clock
(T2CK)<br />  </label>
                    <label><input type="checkbox"   id="ic4"    >  Input Capture 4 (IC4)       <br />
</label>
                    <label><input type="checkbox"   id="ss1"    >  SPI 1 Slave Sync (!SS1)     <br
/>  </label>
                    <label><input type="checkbox"   id="refclk1">  REFCLK1                   <br />
</label>

                    <label><input type="checkbox"   id="int3"   >  External Interrupt 3 (INT3) <br
/>  </label>
                    <label><input type="checkbox"   id="t3ck"   >  Timer3 External Clock
(T3CK)<br />  </label>
                    <label><input type="checkbox"   id="ic3"    >  Input Capture 3 (IC3)       <br />
</label>
                    <label><input type="checkbox"   id="u1cts"  >  !U1CTS                    <br />
</label>
                    <label><input type="checkbox"   id="u2rx"   >  UART 2 Receive (U2RX)
<br />  </label>
                    <label><input type="checkbox"   id="sdi1"   >  SPI 1 In (SDI1)           <br />
</label>

                    <label><input type="checkbox"   id="int2"   >  External Interrupt 2 (INT2) <br
/>  </label>
                    <label><input type="checkbox"   id="t4ck"   >  Timer4 External Clock
(T4CK)<br />  </label>
                    <label><input type="checkbox"   id="ic1"    >  Input Capture 1 (IC1)       <br />
</label>
                    <label><input type="checkbox"   id="ic5"    >  Input Capture 5 (IC5)       <br />
</label>
                    <label><input type="checkbox"   id="u1rx"   >  UART 1 Receive (U1RX)
<br />  </label>
                    <label><input type="checkbox"   id="u2cts"  >  !U2CTS                    <br />
</label>
                    <label><input type="checkbox"   id="sdi2"   >  SPI 2 In (SDI2)           <br />
</label>
                    <label><input type="checkbox"   id="ocfb"   >  OCFB                      <br />
</label>
```

```
                <label><input type="checkbox"   id="int1"   >  External Interrupt 1 (INT1) <br
/>  </label>
                <label><input type="checkbox"   id="t5ck"   >  Timer5 External Clock
(T5CK)<br />  </label>
                <label><input type="checkbox"   id="ic2"    >  Input Capture 2 (IC2)       <br />
</label>
                <label><input type="checkbox"   id="ss2"    >  SPI 2 Slave Sync (!SS2)     <br
/>  </label>
                <label><input type="checkbox"   id="ocfa"   >  OCFA                      <br />
</label>
                </form>

                <form id="outputForm">
                <h4>Selcet output peripherals</h4><br />

                <label><input type="checkbox"   id="u1tx"   >  UART 1 Transmit (U1TX) <br
/>     </label>
                <label><input type="checkbox"   id="u2rts_n">  !U2RTS <br />
</label>
                <label><input type="checkbox"   id="ss1_o"  >  SPI 1 Slave Sync (SS1) <br />
</label>
                <label><input type="checkbox"   id="oc1"    >  Output Compare 1 (OC1) <br
/>     </label>
                <label><input type="checkbox"   id="c2out"  >  Comparator 2 Out (C2OUT)
<br />   </label>


                <label><input type="checkbox"   id="sdo1"   >  SPI 1 Data Out (SDO1) <br />
</label>
                <label><input type="checkbox"   id="sdo2"   >  SPI 2 Data Out (SDO2) <br />
</label>
                <label><input type="checkbox"   id="oc2"    >  Output Compare 2 (OC2) <br
/>      </label>
                <label><input type="checkbox"   id="c3out"  >  Comparator 3 Out (C3OUT)
<br />    </label>


                <label><input type="checkbox"   id="oc4"    >  Output Compare 4 (OC4) <br
/>       </label>
                <label><input type="checkbox"   id="oc5"    >  Output Compare 5 (OC5) <br
/>       </label>
                <label><input type="checkbox"   id="refclko">  REFCLKO <br />
</label>
```

```
                    <label><input type="checkbox"   id="u1rts_n">  !U1RTS <br />
</label>
                    <label><input type="checkbox"   id="u2tx"   >  UART 2 Transmit (U2TX) <br
/>        </label>
                    <label><input type="checkbox"   id="ss2_n"  >  !SPI 2 Slave Sync (!SS2) <br
/>      </label>
                    <label><input type="checkbox"   id="oc3"    >  Output Compare 3 (OC3)<br />
</label>
                    <label><input type="checkbox"   id="c1out"  >  Comparator 1 Out (C1OUT)
<br />      </label>

                    <br /><br /><br /><br />
                    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp<button type="button" id="submit">
&nbsp Submit &nbsp </button> &nbsp&nbsp
                    <button type="reset" id="reset">&nbsp Reset &nbsp</button><br /><br />
                    </form>
      </div>
    </div>
  </div>


  <div class="view hide"  style="display:none" id="site_content2">
    <div class="content">
      <h1>Assignment or Error Message</h1>
      <div class="content_item">
                    <p id="message"></p>

                    </div>
    </div>

  </div>

  <div class="view hide"  style="display:none" id="site_content3">
    <div class="content">
      <h1>Peripherals to Pins Mapping Info</h1>
      <div class="content_item">
                    <table>
                      <tr>
                            <th>Input Peripherals</th>
                            <th>Output Peripherals</th>
                            <th>Pins</th>
                      </tr>
                      <tr>
                            <td>INT4, T2CK, IC4, !SS1, REFCLK1</td>
                            <td>U1TX, !U2RTS, SS1, OC1, C2OUT</td>
                    <td>Pin2, Pin7, Pin11, Pin16, Pin26</td>
                </tr>
                <tr>
                    <td>INT3, T3CK, IC3, !U1CTS, U2RX, SDI1</td>
                    <td>SDO1, SDO2, OC2, C3OUT</td>
                    <td>Pin3, Pin5, Pin14, Pin17, Pin22</td>
                </tr>
                <tr>
                    <td>INT2, T4CK, IC1, IC5, U1RX, !U2CTS, SDI2, OCFB</td>
                    <td>SDO1, SDO2, OC4, OC5, REFCLKO</td>
                    <td>Pin6, Pin9, Pin12, Pin15, Pin24</td>
                </tr>
                <tr>
                    <td>INT1, T5CK, IC2, !SS2, OCFA</td>
                    <td>!U1RTS, U2TX, !SS2, OC3, C1OUT</td>
                    <td>Pin4, Pin10, Pin18, Pin21, Pin25</td>
                </tr>
                </table>
                <p>*Note: peripherals can be mapped to any pin on the same line.</p>

                </div>
      </div>

      <div class="content">
        <h1>Pin Layout</h1>
        <div class="content_item">
                <img src="pinLayout.jpg" alt="Pin layout">
                </div>
      </div>

    </div>


    <footer>
    </footer>
  </div>

  <script src="pps.js"></script>


</body>
</html>
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## JavaScript file

```
///////////////////////MODEL////////////////////////////
var model = {
        message: "", //assignment or error message
        groups: [
                //group 0
                {
                        input: {int4:-1, t2ck: -1, ic4:-1, ss1:-1, refclk1:-1}, //object used to hold
the checked or not checked state of input peripherals in group 1
                        inputPeripheral: ["int4", "t2ck", "ic4", "ss1", "refclk1"],
                        inputChecked: [], //array to hold the checked peripherals
                        output: {u1tx:-1, u2rts_n:-1, ss1_o:-1, oc1:-1, c2out:-1},
                        outputPeripheral: ["u1tx", "u2rts_n", "ss1_o", "oc1", "c2out"],
                        outputPeripheralValue: {u1tx:"0x0001", u2rts_n:"0x0002",
ss1_o:"0x0003", oc1:"0x0005", c2out:"0x0007"}, //RPnR value to OUTPUT peripherals
                        outputChecked: [],
                        pin: {pin2: 1, pin7: 1, pin11: 1, pin16: 1, pin26: 1},
                        pinName: ["pin2", "pin7", "pin11", "pin16", "pin26"],
                        pinReg: {pin2:"RPA0", pin7:"RPB3", pin11:"RPB4", pin16:"RPB7",
pin26:"RPB15"}, //pin,register name dictionary
                        pinValue: {pin2:"0x0000", pin7:"0x0001", pin11:"0x0002",
pin16:"0x0004", pin26:"0x0003"}, //pin value for INPUT peripherals
                        pinLeft: []
                },
                //group 1
                {
                        input: {int3: -1, t3ck: -1, ic3: -1, u1cts: -1, u2rx: -1, sdi1: -1},
                        inputPeripheral: ["int3", "t3ck", "ic3", "u1cts", "u2rx", "sdi1"],
                        inputChecked: [],
                        output: {oc2: -1, c3out: -1},
                        outputPeripheral: ["oc2", "c3out"],
                        outputPeripheralValue: {oc2: "0x0005", c3out: "0x0007"},
                        outputChecked: [],
                        pin: {pin3: 1, pin5: 1, pin14: 1, pin17: 1, pin22: 1},
                        pinName: ["pin3", "pin5", "pin14", "pin17", "pin22"],
                        pinReg: {pin3: "RPA1", pin5: "RPB1", pin14: "PRB5", pin17: "RPB8",
pin22: "RPB11"},
                        pinValue: {pin3: "0x0000", pin5: "0x0002", pin14: "0x0001", pin17:
"0x0004", pin22: "0x0003"},
                        pinLeft: []
                },
                //group 2
                {
                        input: {int2: -1, t4ck: -1, ic1: -1, ic5: -1, u1rx: -1, u2cts: -1, sdi2: -1, ocfb:
-1},
                        inputPeripheral: ["int2", "t4ck", "ic1", "ic5", "u1rx", "u2cts", "sdi2",
"ocfb"],
                        inputChecked: [],
                        output: {oc4: -1, oc5: -1, refclko: -1},
                        outputPeripheral: ["oc4", "oc5", "refclko"],
                        outputPeripheralValue: {oc4: "0x0005", oc5: "0x0006", refclko:
"0x0007"},
                        outputChecked: [],
                        pin: {pin6: 1, pin9: 1, pin12: 1, pin15: 1, pin24: 1},
                        pinName: ["pin6", "pin9", "pin12", "pin15", "pin24"],
                        pinReg: {pin6: "RPB2", pin9: "RPA2", pin12: "RPA4", pin15: "RPB6",
pin24: "RPB13"},
                        pinValue: {pin6: "0x0004", pin9: "0x0000", pin12: "0x0002", pin15:
"0x0001", pin24: "0x0003"},
                        pinLeft: []
                },
                //group 3
                {
                        input: {int1: -1, t5ck: -1, ic2: -1, ss2: -1, ocfa: -1},
                        inputPeripheral: ["int1", "t5ck", "ic2", "ss2", "ocfa"],
                        inputChecked: [],
                        output: {u1rts_n: -1, u2tx: -1, ss2_n: -1, oc3: -1, c1out: -1},
                        outputPeripheral: ["u1rts_n", "u2tx", "ss2_n", "oc3", "c1out"],
                        outputPeripheralValue: {u1rts_n: "0x0001", u2tx: "0x0002", ss2_n:
"0x0004", oc3: "0x0005", c1out: "0x0007"},
                        outputChecked: [],
                        pin: {pin4: 1, pin10: 1, pin18: 1, pin21: 1, pin25: 1},
                        pinName: ["pin4", "pin10", "pin18", "pin21", "pin25"],
                        pinReg: {pin4: "RPB0", pin10: "RPA3", pin18: "RPB9", pin21: "RPB10",
pin25: "RPB14"},
                        pinValue: {pin4: "0x0002", pin10: "0x0000", pin18: "0x0004", pin21:
"0x0003", pin25: "0x0001"},
                        pinLeft: []
                },
                {
                        output: {sdo1: -1, sdo2: -1},
                        outputPeripheral: ["sdo1", "sdo2"],
                        outputPeripheralValue: {sdo1: "0x0003", sdo2: "0x0004"},
                        outputChecked: []
                }
        ] //end of list groups[]

};
```

```
////////////////////////////////CONTROLLER//////////////////////
var controller = {

        init: function(){

                pinSelectionView.init();
                model.message = "";
        },

        flip: function(name, array){ //select or deselect a peripheral or a pin
                array[name] = - array[name];
        },

        checkAllPins: function(){
                var i, j, temp;
                for (i = 0; i <= 3; i++){
                        temp = model.groups[i];
                        for (j = 0; j < temp.pinName.length; j++){
                                temp.pin[temp.pinName[j]] = -1;
                        }
                }
        },

        getCheckedPeripheral: function(){
                var i, j, temp;
                for(i = 0; i <= 3; i++){
                        temp = model.groups[i];

                        temp.inputChecked.splice(0, temp.inputChecked.length);
                        for(j = 0; j < temp.inputPeripheral.length; j++){
                                if(temp.input[temp.inputPeripheral[j]] == 1){
                                        temp.inputChecked.push(temp.inputPeripheral[j]);
                                }
                        }

                        temp.outputChecked.splice(0, temp.outputChecked.length);
                        for(j = 0; j < temp.outputPeripheral.length; j++){
                                if(temp.output[temp.outputPeripheral[j]] == 1){
                                        temp.outputChecked.push(temp.outputPeripheral[j]);
                                }
                        }

                        temp.pinLeft.splice(0, temp.pinName.length);
                for(j = 0; j < temp.pinName.length; j++){
                    if(temp.pin[temp.pinName[j]] == 1){
                        temp.pinLeft.push(temp.pinName[j]);
                    }
                }
            }

            temp = model.groups[4];
            temp.outputChecked.splice(0, temp.outputChecked.length);
            for(j = 0; j <= temp.outputPeripheral.length; j++){
                if(temp.output[temp.outputPeripheral[j]] == 1){
                    temp.outputChecked.push(temp.outputPeripheral[j]);
                }
            }
        },//end of getChecked()

        checkForError: function(){ //return true if there is error
            var error = false;
            var i, temp;
            model.message = "";
            for (i = 0; i <= 3; i++){
                temp = model.groups[i];
                if ( (temp.inputChecked.length + temp.outputChecked.length) >
temp.pinLeft.length ) {
                    error = true;
                    model.message += "<br />" + "Fail to assign ";
                    if (temp.inputChecked.length > 0){
                        model.message +=
temp.inputChecked.length.toString().bold() + "   input peripherals: ".bold() +
temp.inputChecked;
                    }
                    if (temp.outputChecked.length > 0){
                        model.message += " " +
temp.outputChecked.length.toString().bold() +"  output peripherals: ".bold() +
temp.outputChecked;
                    }
                    model.message += " to " + temp.pinLeft.length.toString().bold() +
" pins: ".bold()+ temp.pinLeft;
                }
            }

            //sdo1, sdo2
            if(model.groups[1].pinLeft.length + model.groups[2].pinLeft.length -
(model.groups[1].inputChecked.length + model.groups[1].outputChecked.length +
model.groups[2].inputChecked.length + model.groups[2].outputChecked.length) <
model.groups[4].outputChecked.length ){
                        model.message += "<br /> No enough pins left for " +
model.groups[4].outputChecked + ", which can be mapped to " + model.groups[1].pinName + "
and " + model.groups[2].pinName;
                        error = true;
                }
                if (error == true){
                        model.message += "<br /> <br /> *If possible, think about using similar
peripherals in other groups, go to " + "Reference".bold() + " page to find similar peripherals"

                }
                return error;
        }, //end of checkForError()

        doAssignment: function(){
                if(controller.checkForError()){return;}
                var i, j, checked, pin, length, temp;
                for (i = 0; i <= 3; i++ ){
                        temp = model.groups[i];
                        length = temp.inputChecked.length;
                        if (length > 0){

                                for(j = 0; j < length; j++){
                                        model.message += "<br />";
                                        checked = temp.inputChecked.pop();
                                        pin = temp.pinLeft.pop();
                                        model.message += checked.toUpperCase() +"R" + " = " +
temp.pinValue[pin] + "; //" + "Configure the input " + checked.toUpperCase() + " to pin " +
temp.pinReg[pin] + " (" + pin + ")";
                                }
                        }
                } //end of outer for loop

                model.message += "<br /><br />";
                for (i = 0; i <= 3; i++ ){
                        temp = model.groups[i];
                        length = temp.outputChecked.length;
                        if (length > 0){

                                for (j = 0; j < length; j++){
                                        model.message += "<br />";
                                        checked = temp.outputChecked.pop();
                                        pin = temp.pinLeft.pop();
                                        model.message += temp.pinReg[pin] + "R" + " = " +
temp.outputPeripheralValue[checked] + "; //" + "Configure the output " +
checked.toUpperCase() + " to pin " + temp.pinReg[pin]  + " (" + pin + ")";
                                    }
                              }
                  }

                  //do pin assignment for sdo1 and sdo2
                  temp = model.groups[4];
                  length = temp.outputChecked.length;
                  while(length > 0 && model.groups[1].pinLeft.length > 0){
                        model.message += "<br />";
                        checked = temp.outputChecked.pop();
                        pin = model.groups[1].pinLeft.pop();
                        model.message += model.groups[1].pinReg[pin] + "R" + " = " +
temp.outputPeripheralValue[checked] + "; //" + "Configure the output " +
checked.toUpperCase() + " to pin " + model.groups[1].pinReg[pin] + " (" + pin + ")";
                        length--;
                  }

                  while(length > 0 && model.groups[2].pinLeft.length > 0){
                        model.message += "<br />";
                        checked = temp.outputChecked.pop();
                        pin = model.groups[2].pinLeft.pop();
                        model.message += model.groups[2].pinReg[pin] + "R" + " = " +
temp.outputPeripheralValue[checked] + "; //" + "Configure the output " +
checked.toUpperCase() + " to pin " + model.groups[2].pinReg[pin] + " (" + pin + ")";
                        length--;
                  }

      }//end of doAssignment()

};


///////////////////////////////VIEW/////////////////////////
var pinSelectionView = {

      init: function(){ //add event handlers to all items

                  function sortUnorderedList(ul, sortDescending, nameOfForm) {
                        if (typeof ul == "string") ul = document.getElementById(nameOfForm);

                        var lis = $("#" + nameOfForm+ " label");
                        var vals = [];
```

```
        for (var i = 0, l = lis.length; i < l; i++)
            vals.push(lis[i].innerHTML);
        vals.sort();
        if (sortDescending) vals.reverse();
        for (var i = 0, l = lis.length; i < l; i++)
            lis[i].innerHTML = vals[i];
    }

    function showView(viewName) {
        $('.view').hide();
        $('#' + viewName).show();
    }

    $(document).ready(function(){

        sortUnorderedList("list", false, "inputForm");
        sortUnorderedList("list", false, "outputForm");

        //multiple pages set up
        $('[data-launch-view]').click(function (e) {
            e.preventDefault();
            var viewName = $(this).attr('data-launch-view');
            showView(viewName);
        });


        $('#nav li a').click(function() {
             $('#nav li').removeClass();
             $(this).parent().addClass('current');
        });



        var i, j, temp;
        for(i = 0; i <= 3; i++){
            //add event handler to input peripherals
            temp = model.groups[i];
            for(j = 0; j < temp.inputPeripheral.length; j++){
                $("#"+ temp.inputPeripheral[j]).click((function(name,
array){
                    return function(){
                        controller.flip(name, array);
                    };
                })(temp.inputPeripheral[j], temp.input));
```

```
					}
					//add event handler to output peripherals
					for(j = 0; j < temp.outputPeripheral.length; j++){
						$("#"+ temp.outputPeripheral[j]).click((function(name, 
array){
							return function(){
								controller.flip(name, array);
							};
						})(temp.outputPeripheral[j], temp.output));
					}
					//add event handler to pins
					for(j = 0; j < temp.pinName.length; j++){
						$("#"+ temp.pinName[j]).click((function(name, array){
							return function(){
								controller.flip(name, array);
							};
						})(temp.pinName[j], temp.pin));
					}

				}
				//add even handler for sdo1 and sdo2(the only 2 output peripherals that 
map to two groups of pins)
				temp = model.groups[4];
				for(j = 0; j < temp.outputPeripheral.length; j++){
						$("#"+ temp.outputPeripheral[j]).click((function(name, 
array){
							return function(){
								controller.flip(name, array);
							};
						})(temp.outputPeripheral[j], temp.output));
				}

				$("#checkAllPins").click(function(){
					controller.checkAllPins();
					$("#pinForm input").attr("checked",true);
				});

				$("#submit").click(function(){
					controller.getCheckedPeripheral();
					controller.doAssignment();
					$("#message").html(model.message);
					showView("site_content2");
					$('#nav li').removeClass();
					$("#assignment").addClass('current');
				});
```

```
                        $(":checkbox").attr("autocomplete", "off");
                        $("#reset").click(function(){
                                $("input[type='checkbox']").attr("checked",false);
                                history.go(0);
                        });

                });//end of ready()
        }//end of init()

};


controller.init();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

## CSS file

```
@font-face {
 font-family: Yanone Kaffeesatz;
   src: url('../fonts/YanoneKaffeesatz-Regular.eot');
   src: local(Yanone Kaffeesatz), url('../fonts/YanoneKaffeesatz-Regular.ttf');
}

html {
  height: 100%;}

* {
  margin: 0;
  padding: 0;}

/* tell the browser to render HTML 5 elements as block */
footer, header,nav {
  display:block;}

body {
  font: normal .80em arial, sans-serif;
  background: #F8F7F0;
  color: #7A7A68;
}

p {
  padding: 0 0 20px 0;
  line-height: 1.7em;
}
```

```
img {
  border: 0;
}

h1, h2, h4{
  color: #362C20;
  letter-spacing: 0em;
  padding: 0 0 5px 0;
}

h1, h2{
  font: normal 140% 'Yanone Kaffeesatz', arial;
  margin: 0 0 15px 0;
  padding: 15px 0 5px 0;
  color: #000;
  text-shadow: 1px 1px 1px #fff;
}

h2 {
  font-size: 160%;
  padding: 9px 0 5px 0;
  color: #AEB002;
}


h4{
  color: #AEB002;
  padding: 0 0 5px 0;
  font: normal 110% arial;
  text-transform: uppercase;
}


a, a:hover {
  outline: none;
  text-decoration: underline;
  color: #03D1FD;
}

a:hover {
  text-decoration: none;
}
```

```
ul {
  margin: 2px 0 22px 17px;
}

ul li {
  list-style-type: circle;
  margin: 0 0 6px 0;
  padding: 0 0 4px 5px;
  line-height: 1.5em;
}

li {
  margin: 0 0 11px 0;
}

#main, #logo, nav, #site_content1,#site_content2, #site_content3,footer {
  margin-left: auto;
  margin-right: auto;
}

header {
  background: #323534 url(../images/back.png) repeat-x;
  height: 153px;
}

#logo {
  width: 800px;
  position: relative;
  height: 95px;
  padding-top: 20px;
  background: transparent;
}

#logo #logo_text {
  position: absolute;
  top: 5px;
  left: 0;
}

#logo h1, #logo h2 {
  font-size: 360%;
  border-bottom: 0;
  text-transform: none;
  margin: 0 0 0 15px;
  text-shadow: none;
```

```
}

#logo_text h1, #logo_text h1 a, #logo_text h1 a:hover {
  padding: 5px 0 0 0;
  color: #FFF;
  text-decoration: none;
}

#logo_text h1 a .logo_colour {
  color: #09D4FF;
}

#logo_text a:hover .logo_colour {
  color: #FFF;
}

#logo_text h2 {
  font-size: 125%;
  padding: 0 0 0 0;
  color: #A8AA94;
}

nav {
  width: 800px;
  height: 46px;
}

#site_content1, #site_content2, #site_content3 {
  width: 800px;
  overflow: hidden;
  margin: 20px auto 0 auto;
  padding: 0 0 10px 0;
}

  .content h1 {
  background: #eee;padding: 10px 15px;
  font-size: 170%;
  border: 1px solid #fff;
  color: #063E4C;
    border-radius: 7px 7px 0 0;
  -moz-border-radius: 7px 7px 0 0;
  -webkit-border: 7px 7px 0 0;
}

.content h1 {
  padding: 5px 15px 10px 15px;
  color: #74706E;
}


.content_item {
  padding: 15px 15px;
}
#inputForm, #outputForm{
          float: left;
          padding: 15px 80px 30px 80px;

}
.content {
  text-align: left;
  width: 800px;
  margin: 0 0 15px 0;
  background: #fff;border-bottom: 1px solid #D4D4D4;
  border: 1px solid #e5e5e5;
  border-radius: 7px 7px 7px 7px;
  -moz-border-radius: 7px 7px 7px 7px;
  -webkit-border: 7px 7px 7px 7px;
  float: left;
}



footer {
  width: 100%;
  height: 60px;
  padding: 20px 0 5px 0;
  border-top: 1px solid #000;
  background: #545454;
  color: #A8AA94;
  margin: 0 0 0 0;
  bottom: 0;
}


table {
  margin: 10px 0 30px 0;
}

table tr th, table tr td {
  background: #3B3B3B;
  color: #FFF;
  padding: 7px 4px;
  text-align: left;
}

table tr td {
  background: #E5E5DB;
  color: #47433F;
  border-top: 1px solid #FFF;
}


/* Configuration of menu width */
html body ul.sf-menu ul, html body ul.sf-menu ul li {
  width: 180px;
}

html body ul.sf-menu ul ul {
  margin: 0 0 0 180px;
}
body {
	margin-bottom: 70px;
}
/* Framework for proper showing/hiding/positioning */
ul.sf-menu, ul.sf-menu * {
  margin: 0;
  padding: 0;
}

ul.sf-menu {
  display: block;
  position: relative;
}

ul.sf-menu li {
  display: block;
  list-style: none;
  float: left;
  position: relative;
}

ul.sf-menu li:hover {
  visibility: inherit; /* fixes IE7 'sticky bug' */
}
```

```
ul.sf-menu a {
  display: block;
  position: relative;
}

ul.sf-menu ul {
  position: absolute;
  left: 0;
  width: 150px;
  top: auto;
  left: -999999px;
}

ul.sf-menu ul a {
  zoom: 1; /* IE6/7 fix */
}

ul.sf-menu ul li {
  float: left; /* Must always be floated otherwise there will be a rogue 1px margin-bottom in
IE6/7 */
  width: 150px;
}

ul.sf-menu ul ul {
  top: 0;
  margin: 0 0 0 150px
}

ul.sf-menu li:hover ul,ul.sf-menu li:focus ul,ul.sf-menu li.sf-hover ul,
ul.sf-menu ul li:hover ul,ul.sf-menu ul li:focus ul,ul.sf-menu ul li.sf-hover ul,
ul.sf-menu ul ul li:hover ul,ul.sf-menu ul ul li:focus ul,ul.sf-menu ul ul li.sf-hover ul,
ul.sf-menu ul ul ul li:hover ul,ul.sf-menu ul ul ul li:focus ul,ul.sf-menu ul ul ul li.sf-hover ul {
  left: auto;
}

ul.sf-menu li:hover ul ul,ul.sf-menu li:focus ul ul,ul.sf-menu li.sf-hover ul ul,
ul.sf-menu ul li:hover ul ul,ul.sf-menu ul li:focus ul ul,ul.sf-menu ul li.sf-hover ul ul,
ul.sf-menu ul ul li:hover ul ul,ul.sf-menu ul ul li:focus ul ul,ul.sf-menu ul ul li.sf-hover ul ul,
ul.sf-menu ul ul ul li:hover ul ul,ul.sf-menu ul ul ul li:focus ul ul,ul.sf-menu ul ul ul li.sf-hover ul
ul {
  left: -999999px;
}

/* autoArrows CSS */
span.sf-arrow {
  width: 7px;
  height: 7px;
  position: absolute;
  top: 8px;
  right: 0;
  display: block;
  background: url(../images/images/arrows-black.png) no-repeat 0 0;
  overflow: hidden; /* making sure IE6 doesn't overflow and expand the box */
  font-size: 1px;
}

ul ul span.sf-arrow {
  right: 12px;
  top: 7px;
  background-position: 0 100%;
}

/* Theming the menu */
ul#nav {
  float: left;
}

ul#nav li.current a {
  color: #FFF;}

ul#nav ul {
  background: #88E2F8;
}

ul#nav li a {
  padding: 7px 20px;
  font-family: 'trebuchet ms',helvetica,arial,verdana,sans;
  text-decoration: none;
  color: #111;
  background: #88E2F8;
  margin-right: 2px;
  border-radius: 7px 7px 0 0;
  -moz-border-radius: 7px 7px 0 0;
  -webkit-border: 7px 7px 0 0;}

ul#nav li a:hover, ul#nav li a:focus {
  color: #FFF;
}

ul#nav li ul li a {
  margin-right: 0;
  border-radius: 0 0 0 0;
  -moz-border-radius: 0 0 0 0;
  -webkit-border: 0 0 0 0;
}

span.sf-arrow {
  top: 15px;
  right: 5px;
  background-image: url(../images/arrows-white.png);
}

ul ul span.sf-arrow {
  right: 12px;
  top: 15px;
}
```